Computer Software: The 'Trojan Horse' of HPC

Co-Designing a COTS Re-configurable Exascale Computer

Steve Wallach
swallach"at"conveycomputer "dot"com
Convey <= Convex++

Acknowledgement

- Steve Poole (ORNL) and I had many emails and discussion on technology trends.
- We are both guilty of being overzealous architects

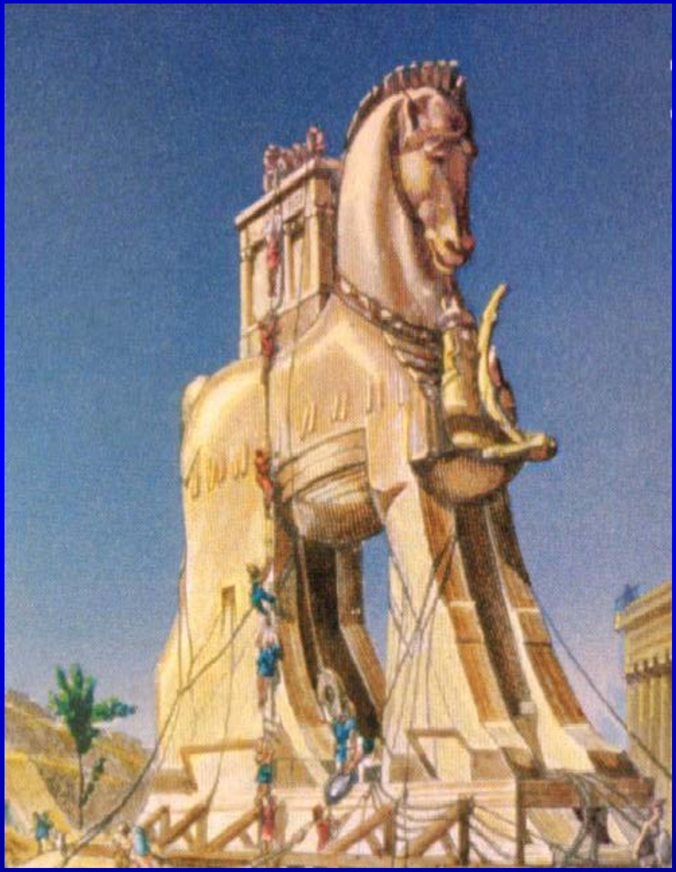
Discussion

- The state of HPC software – today
- The path to Exascale Computing

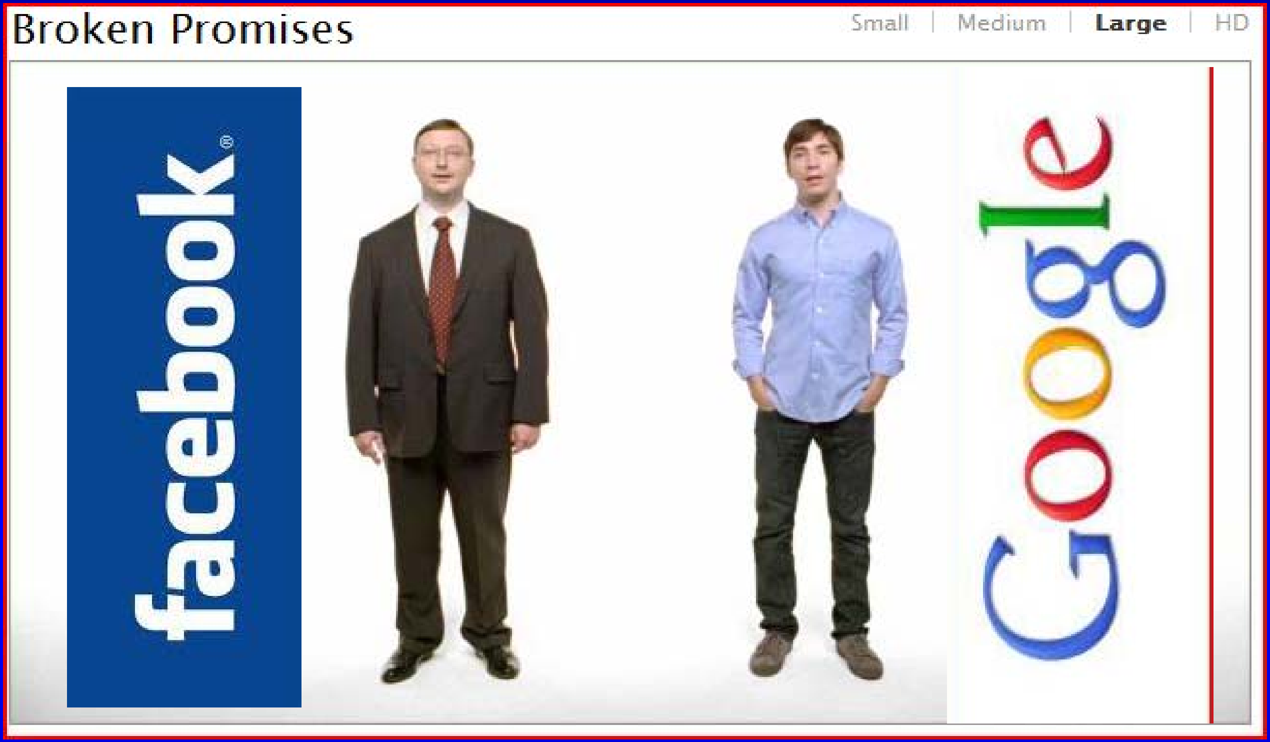
Mythology

- In the old days, we were told

  "Beware of Greeks bearing gifts"



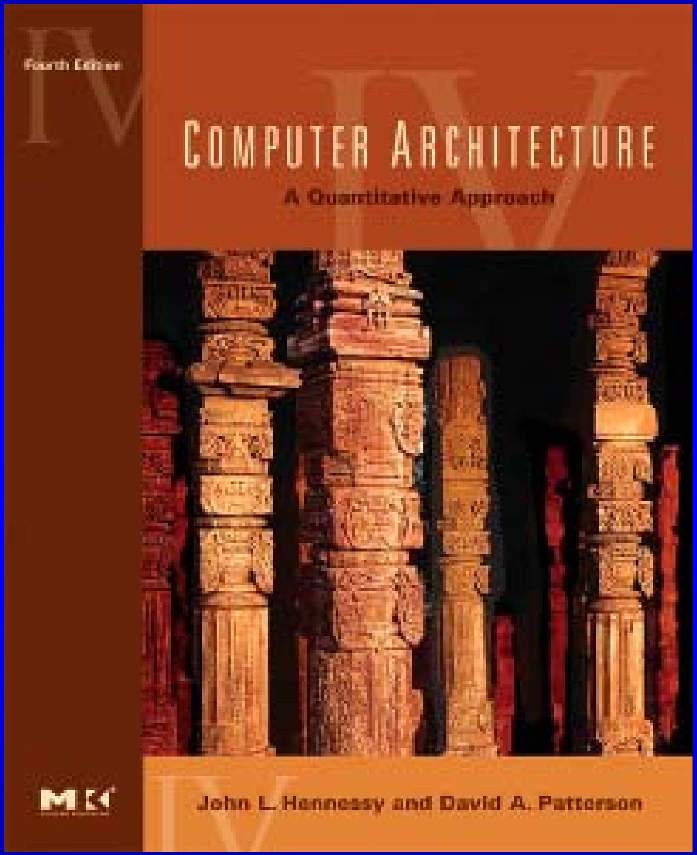
Today



- "Beware of Geeks bearing Gifts"

What problems are we solving

- New Hardware Paradigms
- Uniprocessor Performance leveling
- MPP and multi-threading for the masses

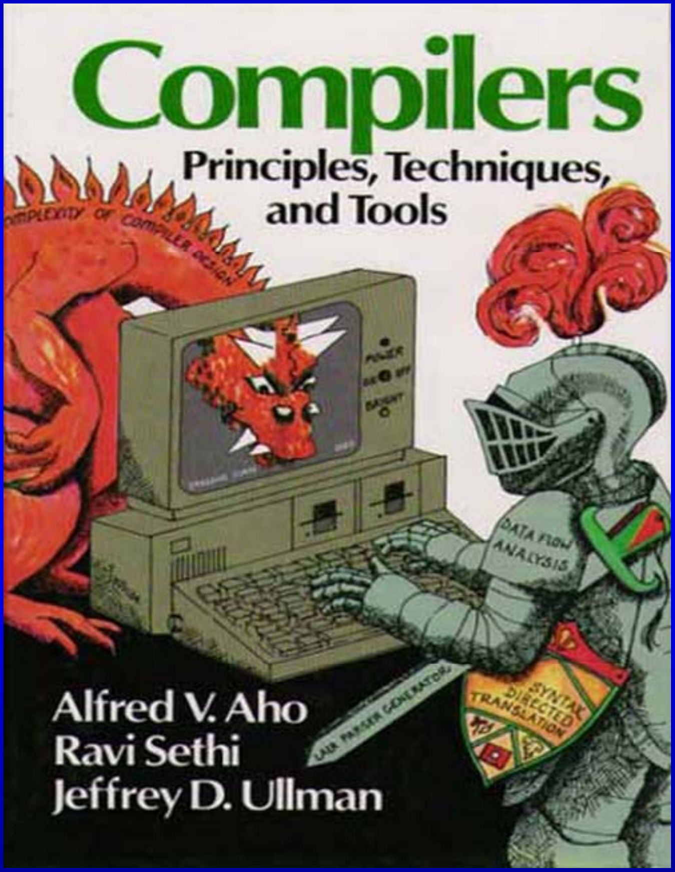


Deja Vu

- Multi-Core Evolves
  - Many Core
  - ILP fizzles
- x86 extended with sse2, sse3, and sse4
  - application specific enhancements
  - Co-processor within x86 micro-architecture
- Basically performance enhancements by
  - On chip parallel
  - Instructions for specific application acceleration
    - One application instruction replaces MANY generic instructions
- Déjà vu – all over again – 1980's
  - Need more performance than micro
  - GPU, CELL, and FPGA's
    - Different software environment
- Heterogeneous Computing AGAIN

Current Languages

- Fortran 66 → Fortran 77 → Fortran 95 → 2003
  - HPC Fortran
  - Co-Array Fortran
- C → C++
  - UPC
  - Stream C
  - C# (Microsoft)
  - Ct (Intel)


Compilers
Principles, Techniques, and Tools
Alfred V. Aho
Ravi Sethi
Jeffrey D. Ullman
COMPLEXITY OF COMPILER DESIGN
DATA FLOW ANALYSIS
SYNTAX DIRECTED TRANSLATION
LALR PARSER GENERATOR

Another Bump in the Road

- GPGPU's are very cost effective for many applications.
- Matrix Multiply
  - Fortran

```
do i = 1,n1
do k = 1,n3
c(i,k) = 0.0
do j = 1,n2
c(i,k) = c(i,k) + a(i,j) * b(j,k)
Enddo
Enddo
Enddo
```

PGI Fortran to CUDA

```
__global__ void
matmulKernel( float* C, float* A, float* B, int N2, int N3 ){
  int bx = blockIdx.x,  by = blockIdx.y;
  int tx = threadIdx.x, ty = threadIdx.y;
  int aFirst = 16 * by * N2;
  int bFirst = 16 * bx;
  float Csub = 0;
  for( int j = 0; j < N2; j += 16 ) {
      shared   float Atile[16][16], Btile[16][16];
    Atile[ty][tx] = A[aFirst + j + N2 * ty + tx];
    Btile[ty][tx] = B[bFirst + j*N3 + b + N3 * ty + tx];
    __syncthreads();
     for( int k = 0; k < 16; ++k )
       Csub += Atile[ty][k] * Btile[k][tx];
       syncthreads();
  }
  int c = N3 * 16 * by + 16 * bx;
  C[c + N3 * ty + tx] = Csub;
}
void
matmul( float* A, float* B, float* C,
             size_t N1, size_t N2, size_t N3 ){
  void *devA, *devB, *devC;
  cudaSetDevice(0);
  cudaMalloc( &devA, N1*N2*sizeof(float) );
  cudaMalloc( &devB, N2*N3*sizeof(float) );
  cudaMalloc( &devC, N1*N3*sizeof(float) );
  cudaMemcpy( devA, A, N1*N2*sizeof(float), cudaMemcpyHostToDevice );
  cudaMemcpy( devB, B, N2*N3*sizeof(float), cudaMemcpyHostToDevice );
  dim3 threads( 16, 16 );
  dim3 grid( N1 / threads.x, N3 / threads.y);
  matmulKernel<<< grid, threads >>>( devC, devA, devB, N2, N3 );
  cudaMemcpy( C, devC, N1*N3*sizeof(float), cudaMemcpyDeviceToHost );
  cudaFree( devA );
  cudaFree( devB );
  cudaFree( devC );
}
```

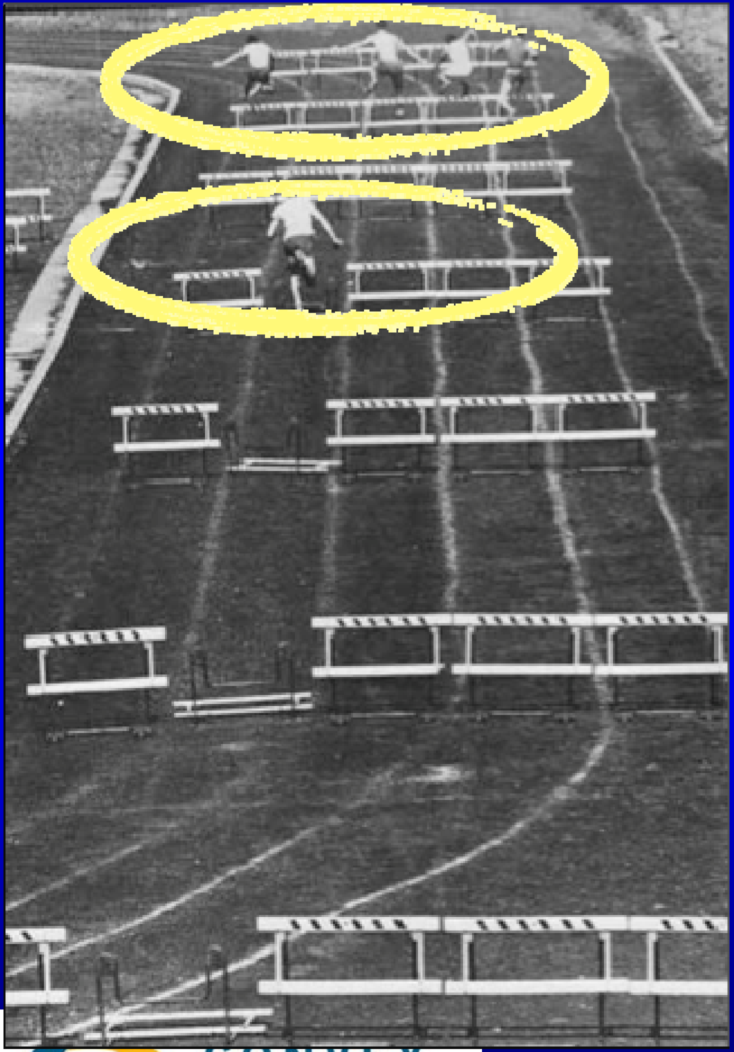
Pornographic Programming:
Can't define it, but you know
When you see it.

http://www.linuxjournal.com/article/10216
Michael Wolfe – Portland Group
How We Should Program GPGPUs November 1st, 2008

CONVEY computer

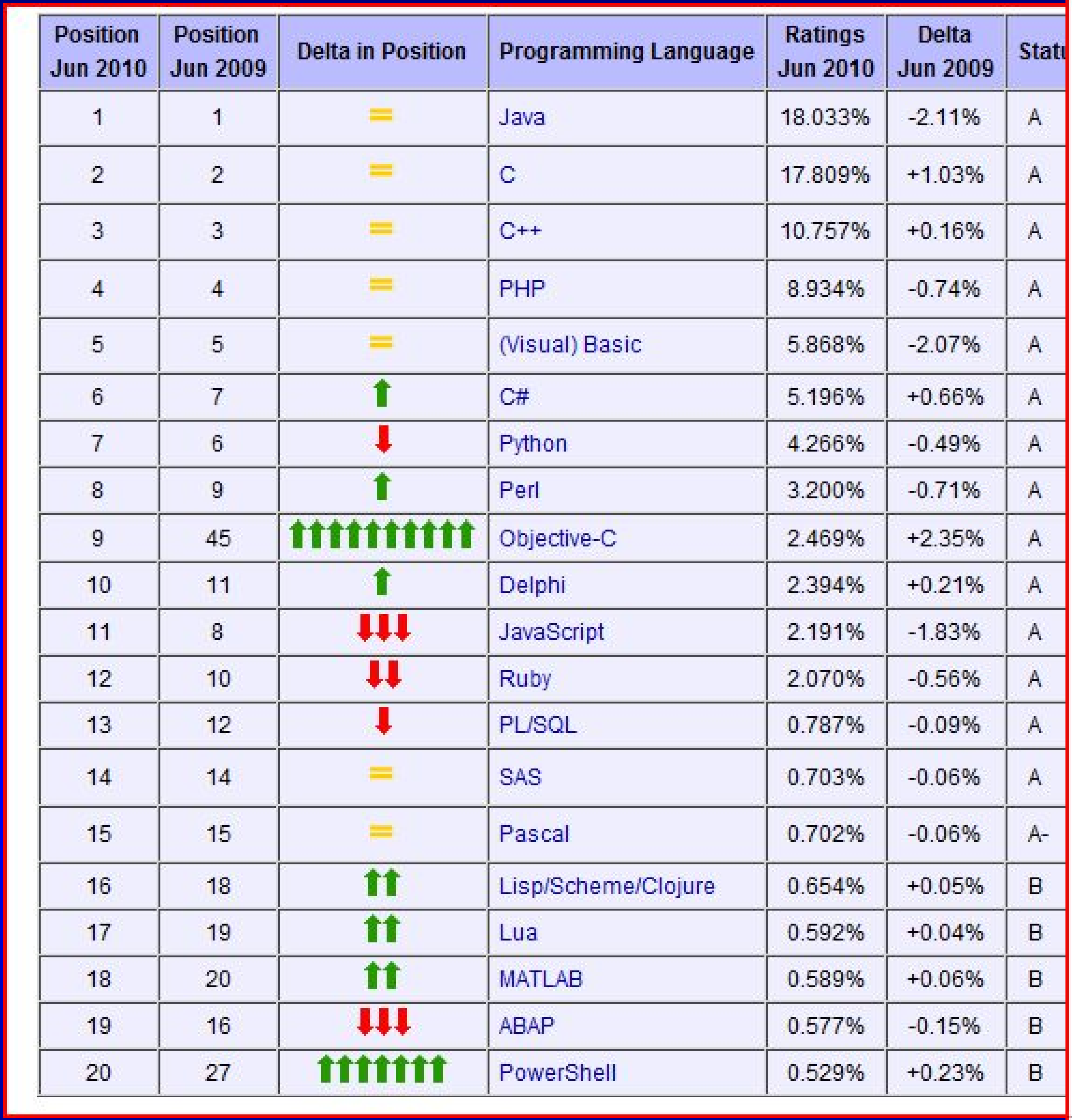
Programmer

~~Find the~~ Accelerator

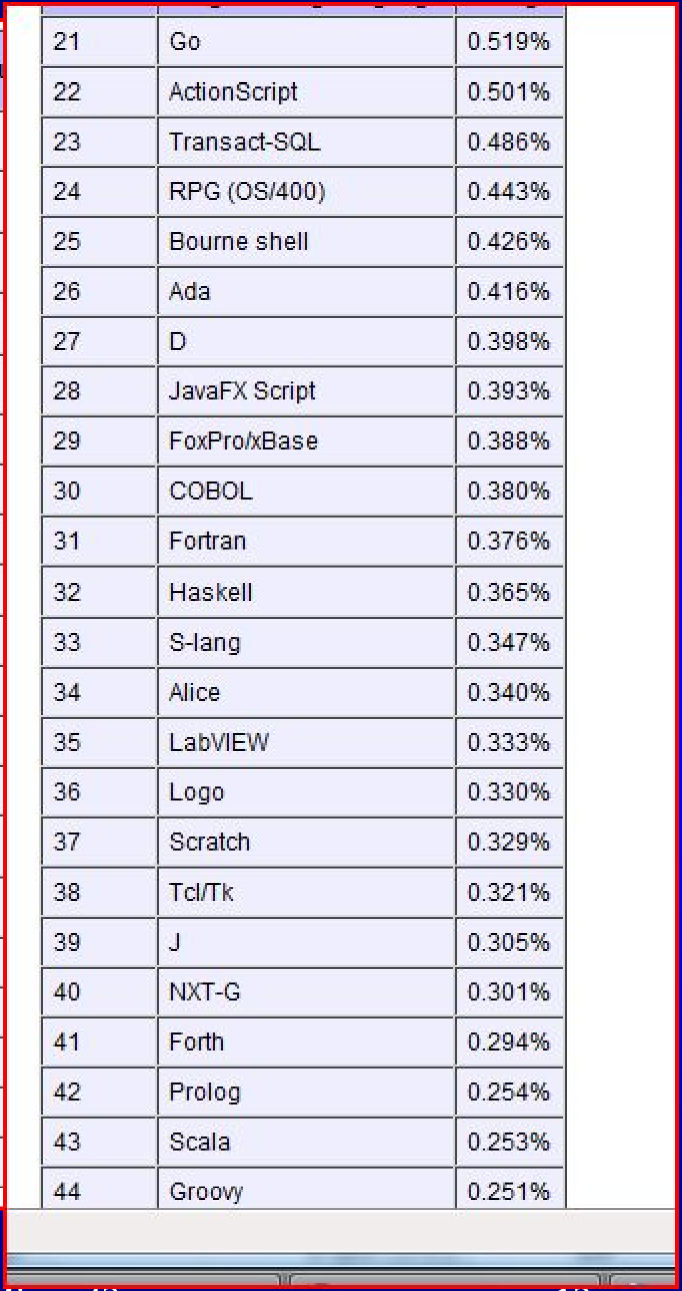


- Accelerators can be beneficial. It isn't "free" (like waiting for the next clock speed boost)
  - Worst case - you will have to completely rethink your algorithms and/or data structures
  - Performance tuning is still time consuming
  - Don't forget our long history of parallel computing...

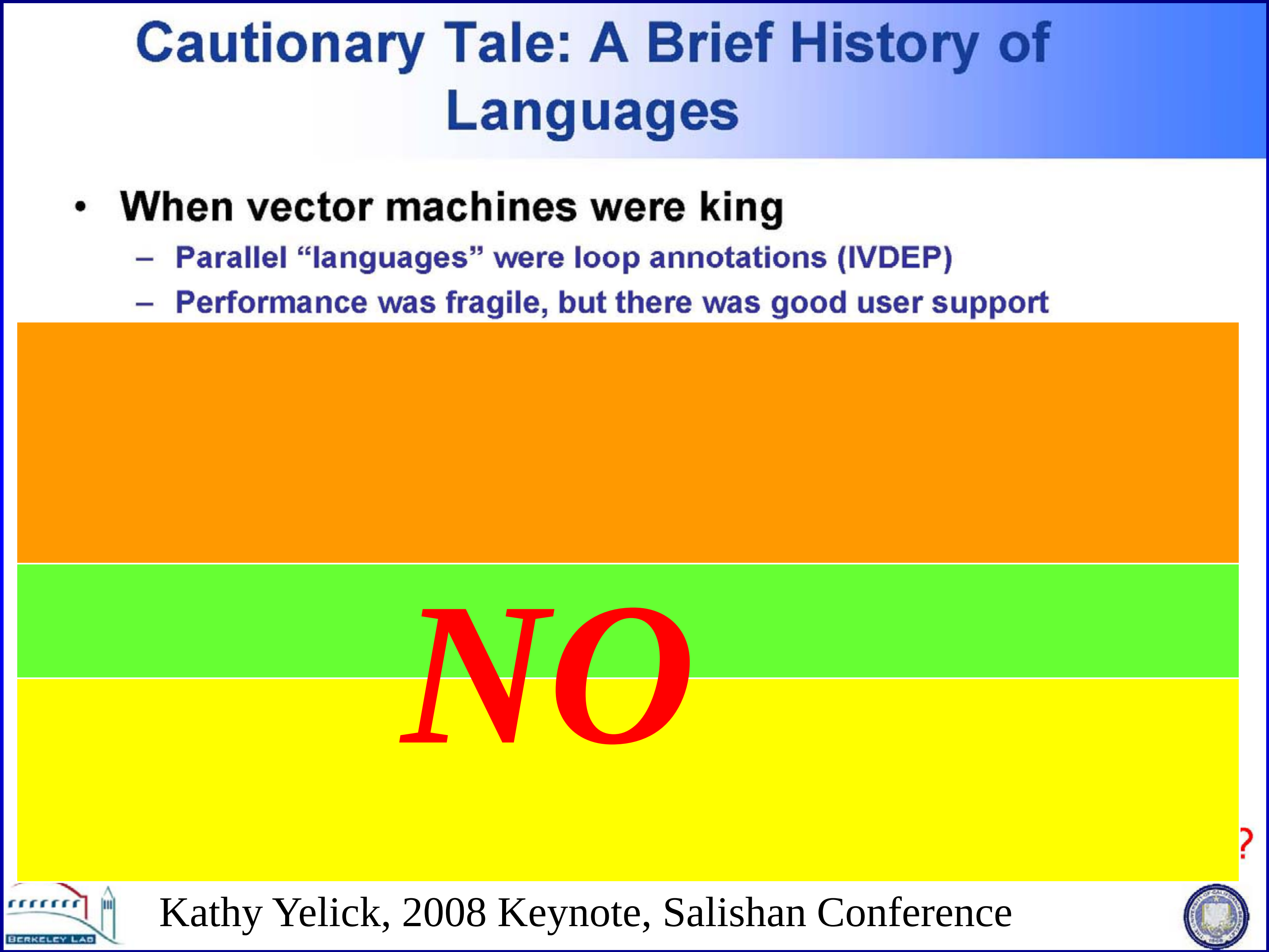
Fall Creek Falls Conference, Chattanooga, TN - Sept. 2009
*One of These Things Isn't Like the Other...Now What?*Pat McCormick, LANL



Position Jun 2010	Position Jun 2009	Delta in Position	Programming Language	Ratings Jun 2010	Delta Jun 2009	Statu
1	1	=	Java	18.033%	-2.11%	A
2	2	=	C	17.809%	+1.03%	A
3	3	=	C++	10.757%	+0.16%	A
4	4	=	PHP	8.934%	-0.74%	A
5	5	=	(Visual) Basic	5.868%	-2.07%	A
6	7	↑	C#	5.196%	+0.66%	A
7	6	↓	Python	4.266%	-0.49%	A
8	9	↑	Perl	3.200%	-0.71%	A
9	45	↑↑↑↑↑↑↑↑↑↑	Objective-C	2.469%	+2.35%	A
10	11	↑	Delphi	2.394%	+0.21%	A
11	8	↓↓↓	JavaScript	2.191%	-1.83%	A
12	10	↓↓	Ruby	2.070%	-0.56%	A
13	12	↓	PL/SQL	0.787%	-0.09%	A
14	14	=	SAS	0.703%	-0.06%	A
15	15	=	Pascal	0.702%	-0.06%	A-
16	18	↑↑	Lisp/Scheme/Clojure	0.654%	+0.05%	B
17	19	↑↑	Lua	0.592%	+0.04%	B
18	20	↑↑	MATLAB	0.589%	+0.06%	B
19	16	↓↓↓	ABAP	0.577%	-0.15%	B
20	27	↑↑↑↑↑↑↑	PowerShell	0.529%	+0.23%	B



21	Go	0.519%
22	ActionScript	0.501%
23	Transact-SQL	0.486%
24	RPG (OS/400)	0.443%
25	Bourne shell	0.426%
26	Ada	0.416%
27	D	0.398%
28	JavaFX Script	0.393%
29	FoxPro/xBase	0.388%
30	COBOL	0.380%
31	Fortran	0.376%
32	Haskell	0.365%
33	S-lang	0.347%
34	Alice	0.340%
35	LabVIEW	0.333%
36	Logo	0.330%
37	Scratch	0.329%
38	Tcl/Tk	0.321%
39	J	0.305%
40	NXT-G	0.301%
41	Forth	0.294%
42	Prolog	0.254%
43	Scala	0.253%
44	Groovy	0.251%



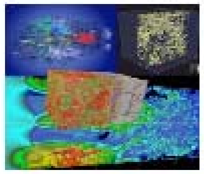
Cautionary Tale: A Brief History of Languages

- **When vector machines were king**
  - Parallel "languages" were loop annotations (IVDEP)
  - Performance was fragile, but there was good user support

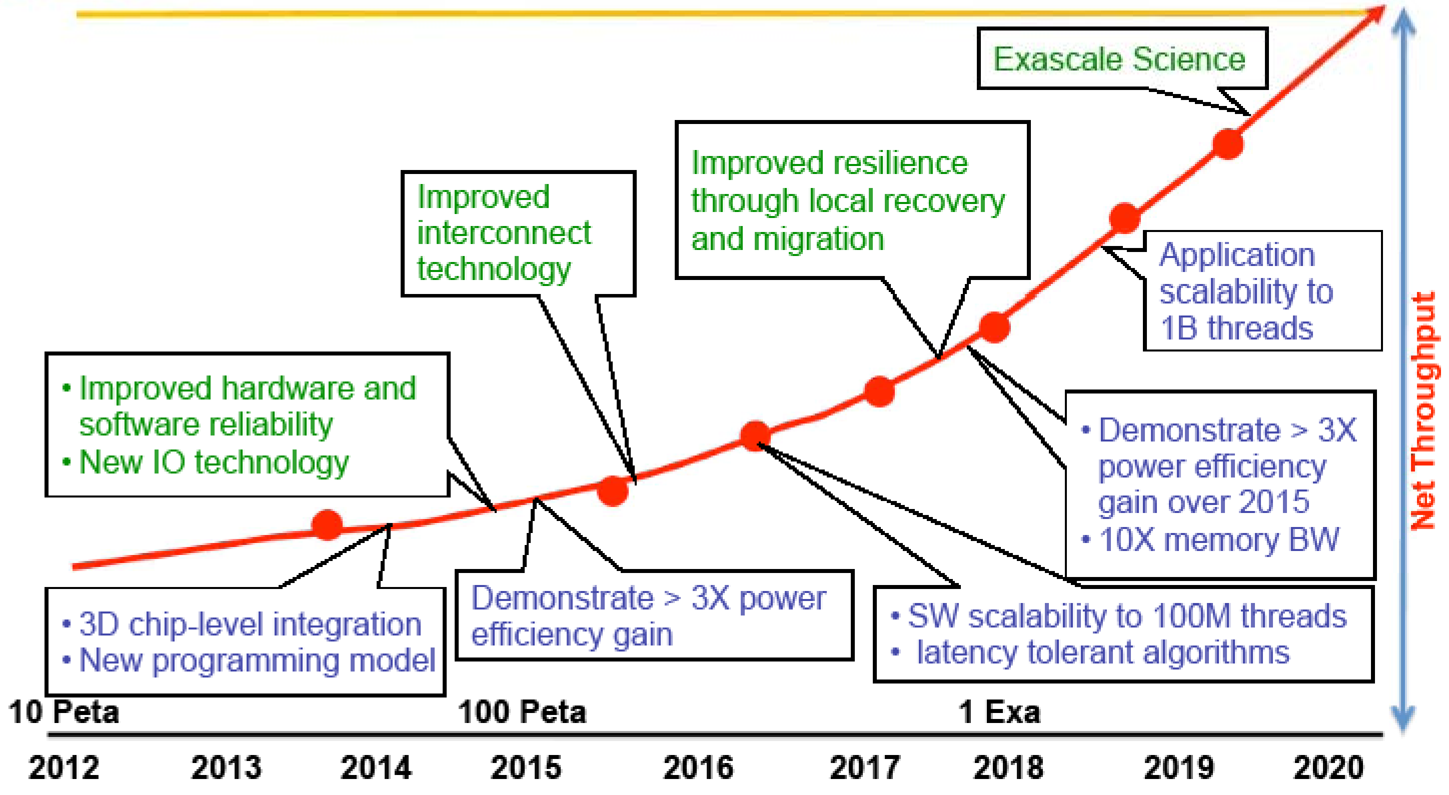
?

NO

Kathy Yelick, 2008 Keynote, Salishan Conference



Technology Roadmap

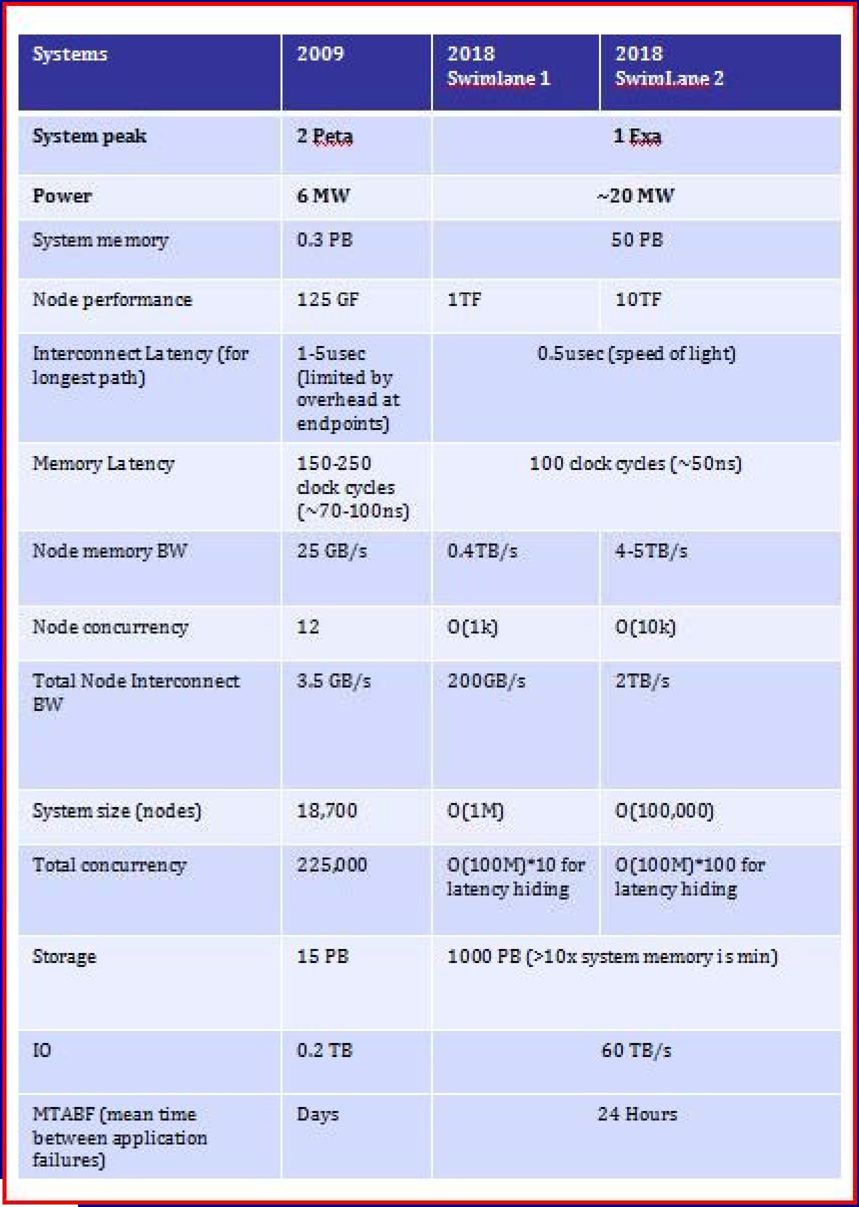


Exascale Science
Improved resilience through local recovery and migration
Improved interconnect technology
Application scalability to 1B threads
• Improved hardware and software reliability
• New IO technology
• Demonstrate > 3X power efficiency gain over 2015
• 10X memory BW
• 3D chip-level integration
• New programming model
Demonstrate > 3X power efficiency gain
• SW scalability to 100M threads
• latency tolerant algorithms
Net Throughput
10 Peta
100 Peta
1 Exa
2012
2013
2014
2015
2016
2017
2018
2019
2020

U.S. DEPARTMENT OF ENERGY

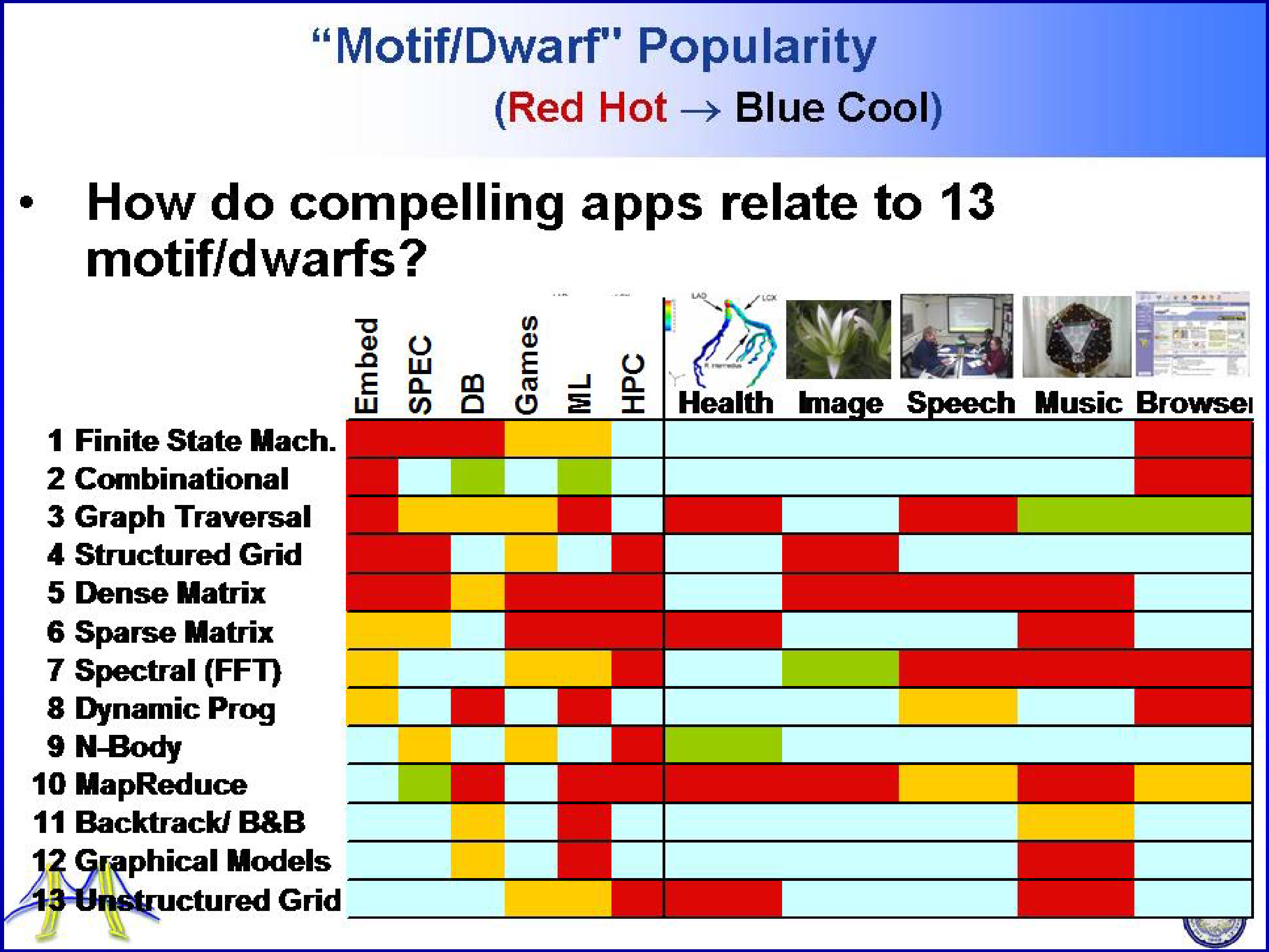
DOE Exascale Initiative Technical Roadmap

Slide 12



Systems	2009	2018 Swimlane 1	2018 SwimLane 2
System peak	**2 Peta**	**1 Exa**	
Power	**6 MW**	**~20 MW**	
System memory	0.3 PB	50 PB	
Node performance	125 GF	1TF	10TF
Interconnect Latency (for longest path)	1-5usec (limited by overhead at endpoints)	0.5usec (speed of light)	
Memory Latency	150-250 clock cycles (~70-100ns)	100 clock cycles (~50ns)	
Node memory BW	25 GB/s	0.4TB/s	4-5TB/s
Node concurrency	12	O(1k)	O(10k)
Total Node Interconnect BW	3.5 GB/s	200GB/s	2TB/s
System size (nodes)	18,700	O(1M)	O(100,000)
Total concurrency	225,000	O(100M)*10 for latency hiding	O(100M)*100 for latency hiding
Storage	15 PB	1000 PB (>10x system memory is min)	
IO	0.2 TB	60 TB/s	
MTABF (mean time between application failures)	Days	24 Hours	

Berkeley's 13 Motifs



"Motif/Dwarf" Popularity

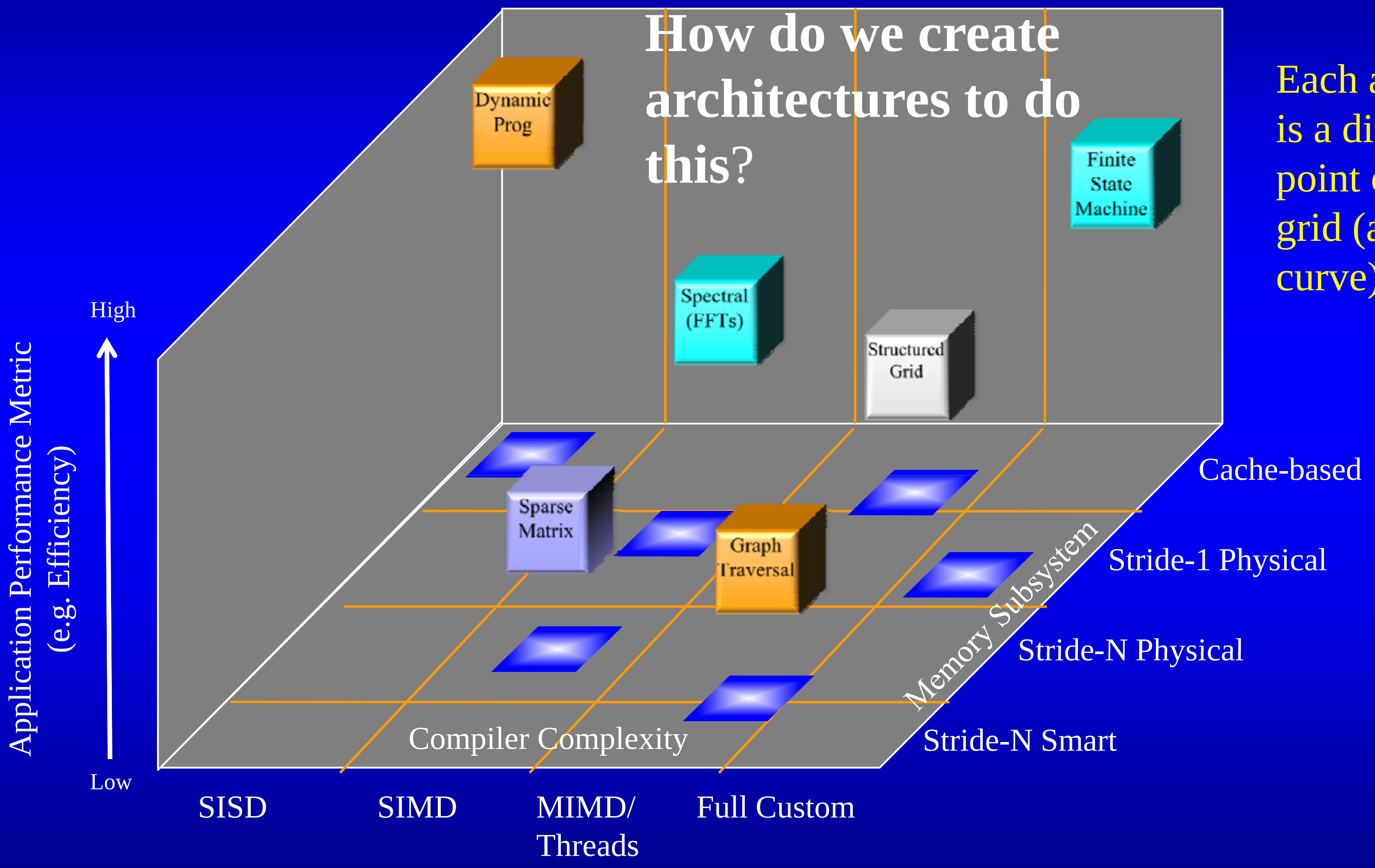
(Red Hot → Blue Cool)

- How do compelling apps relate to 13 motif/dwarfs?

	Embed	SPEC	DB	Games	ML	HPC	Health	Image	Speech	Music	Browser
1 Finite State Mach.	🟥	🟥	🟥	🟨	🟨	🟦	🟦	🟦	🟦	🟦	🟥
2 Combinational	🟥	🟦	🟩	🟦	🟩	🟦	🟦	🟦	🟦	🟦	🟥
3 Graph Traversal	🟥	🟨	🟨	🟨	🟥	🟦	🟥	🟦	🟥	🟩	🟩
4 Structured Grid	🟥	🟥	🟦	🟨	🟦	🟥	🟦	🟥	🟦	🟦	🟦
5 Dense Matrix	🟥	🟥	🟨	🟥	🟥	🟥	🟦	🟥	🟥	🟥	🟦
6 Sparse Matrix	🟨	🟨	🟦	🟥	🟥	🟥	🟥	🟦	🟦	🟥	🟦
7 Spectral (FFT)	🟨	🟦	🟦	🟨	🟨	🟥	🟦	🟩	🟥	🟥	🟥
8 Dynamic Prog	🟨	🟦	🟥	🟦	🟥	🟦	🟦	🟦	🟨	🟦	🟥
9 N-Body	🟦	🟨	🟦	🟨	🟦	🟥	🟩	🟦	🟦	🟦	🟦
10 MapReduce	🟦	🟩	🟥	🟦	🟥	🟥	🟥	🟥	🟨	🟥	🟨
11 Backtrack/ B&B	🟦	🟦	🟨	🟦	🟥	🟦	🟦	🟦	🟦	🟨	🟦
12 Graphical Models	🟦	🟦	🟨	🟦	🟥	🟦	🟦	🟦	🟦	🟥	🟦
13 Unstructured Grid	🟦	🟦	🟦	🟨	🟨	🟥	🟥	🟦	🟦	🟥	🟦

http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-23.html

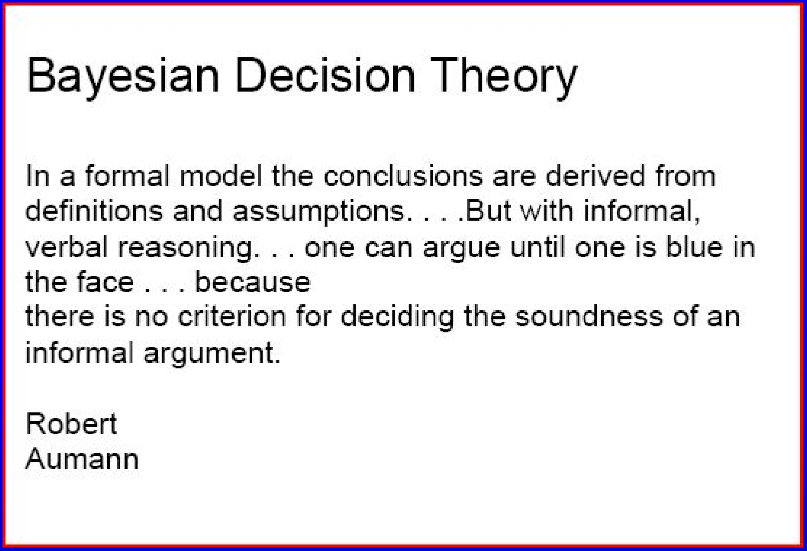
What does this all mean?



Each application is a different point on this 3D grid (actually a curve)

Take an Operation Research Method of Prediction

- Moore's Law
- Application Specific
  - Matrices
  - Matrix Arithmetic
- Hard IP for floating point
- Number of Arithmetic Engines
- System Arch and Programming model
  - Language directed design
- Resulting Analysis
  - Benefit
  - Mean and +/- sigma (if normal)



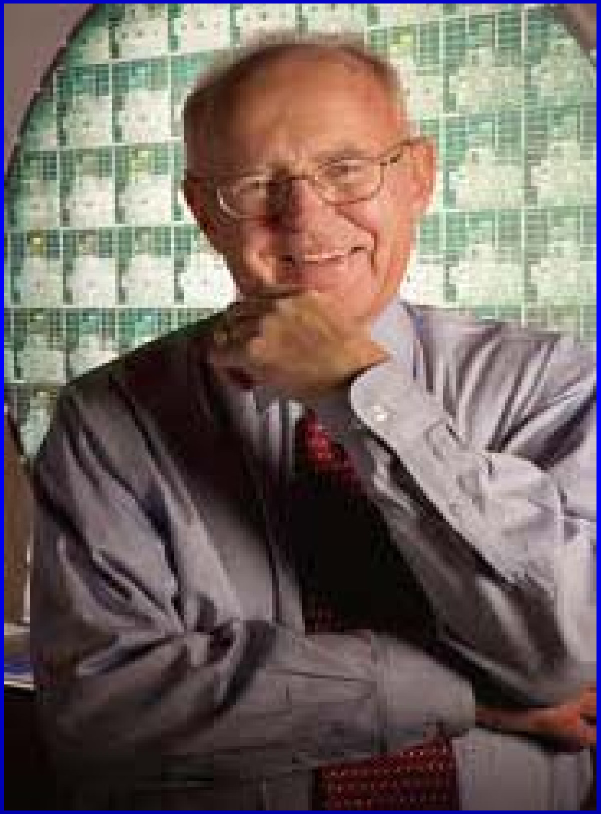
Bayesian Decision Theory

In a formal model the conclusions are derived from definitions and assumptions. . . .But with informal, verbal reasoning. . . one can argue until one is blue in the face . . . because
there is no criterion for deciding the soundness of an informal argument.

Robert
Aumann

Moore’s Law

- Feature Set
  - Every 2 years twice the logic
  - Thus by 2020
    - 8 times the logic, same clock rate
  - Mean Factor of 7, sigma +/- 2
- Benefit
  - 7 times the performance, same clock rate, same internal architecture



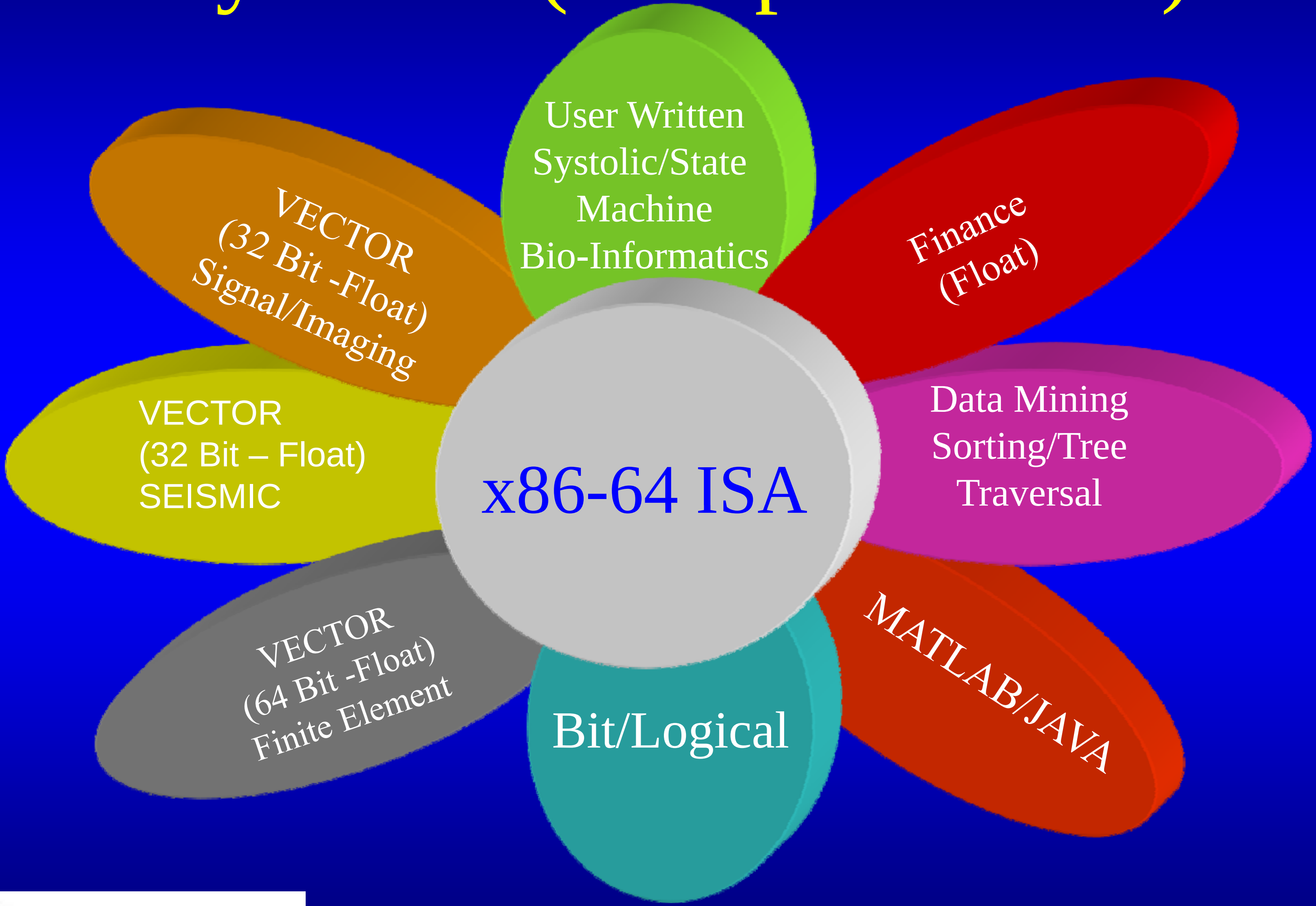
Rock's Law

- **Rock's law**, named for Arthur Rock, says that the cost of a semiconductor chip fabrication plant doubles every four years. As of 2003, the price had already reached about 3 billion US dollars.

- Rock's Law can be seen as the economic flipside to Moore's Law; the latter is a direct consequence of the ongoing growth of the capital-intensive semiconductor industry—innovative and popular products mean more profits, meaning more capital available to invest in ever higher levels of large-scale integration, which in turn leads to creation of even more innovative products.

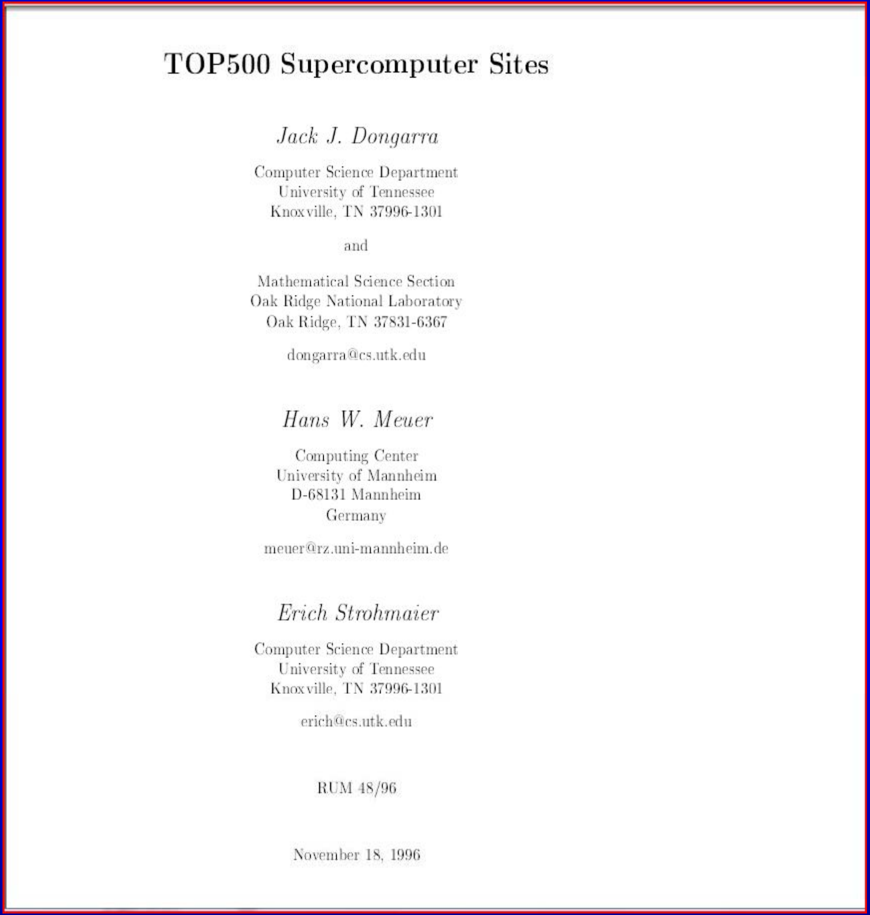
http://en.wikipedia.org/wiki/Rock%27s_law

Convey – ISA (Compiler View)



Application Specific

- Feature Set
  - Matrix Engine
  - Mean Factor of 4 (+4/- 1)
- June 1993 Linpack (% of peak)
  - NEC SX3/44 - 95% (4)
  - Cray YMP (16) – 90% (9)
- June 2006
  - Earth Simulator (NEC) – 87.5% (5120)
- June 2010 Linpack
  - Jaguar - 75% (224162 cores & GPU)
- November 2010 Linpack
  - Tianhe-1A – 53% (86016 cores & GPU)
  - French Bull – 83% (17,480 sockets. 140k cores)
- Benefit
  - 90% of Peak
  - Matrix Arithmetic's
  - Outer Loop Parallel/Inner Loop vector within ONE functional unit



TOP500 Supercomputer Sites

Jack J. Dongarra

Computer Science Department
University of Tennessee
Knoxville, TN 37996-1301

and

Mathematical Science Section
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6367

dongarra@cs.utk.edu

Hans W. Meuer

Computing Center
University of Mannheim
D-68131 Mannheim
Germany

meuer@rz.uni-mannheim.de

Erich Strohmaier

Computer Science Department
University of Tennessee
Knoxville, TN 37996-1301

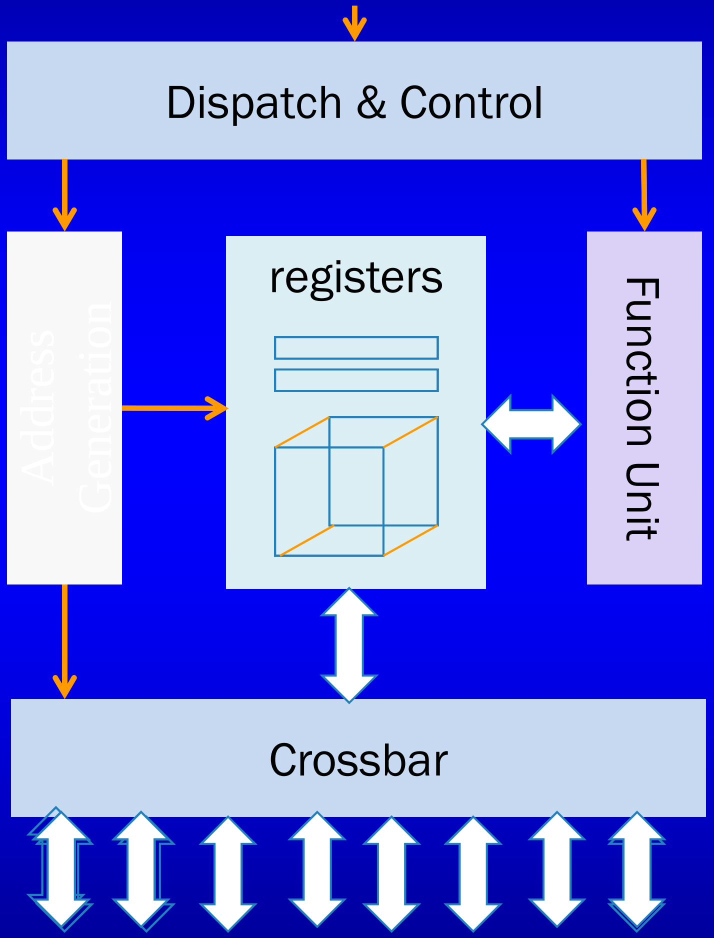
erich@cs.utk.edu

RUM 48/96

November 18, 1996

3D

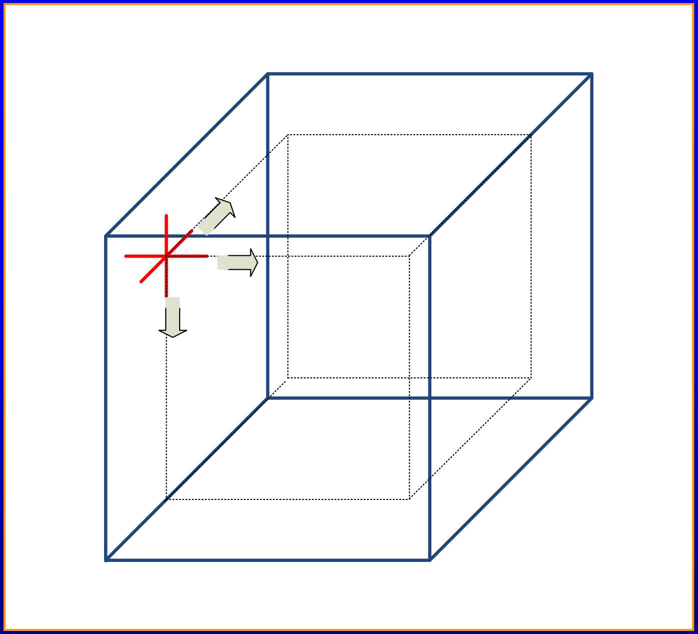
- Modular content
  - Datatypes: IEEE SP, IEEE DP, 32-bit integer, 64-bit integer, bit
  - Operations: simple, compound, reductions
  - Dimensionality: 1d, 2d, 3d
  - memory access: gather/scatter, strided, under mask, multidimensional
  - register access pattern: halo, sparse periodic access, under mask, multidimensional
- Number of function units determined by functionality selected



3D Finite Difference (3DFD) Personality

- designed for nearest neighbor operations on structured grids
  - maximizes data reuse
- reconfigurable "registers"
  - 1D (vector), 2D, and 3D modes
  - 8192 elements in a register
- operations on entire cubes
  - "add points to their neighbor to the left times a scalar" is a single instruction
  - up to 7 points away in any direction
- finite difference method for post-stack reverse-time migration

```
X(I,J,K) = S0*Y(I  ,J  ,K  )
         + S1*Y(I-1,J  ,K  )
         + S2*Y(I+1,J  ,K  )
         + S3*Y(I  ,J-1,K  )
         + S4*Y(I  ,J+1,K  )
         + S5*Y(I  ,J  ,K-1)
         + S6*Y(I  ,J  ,K+1)
```



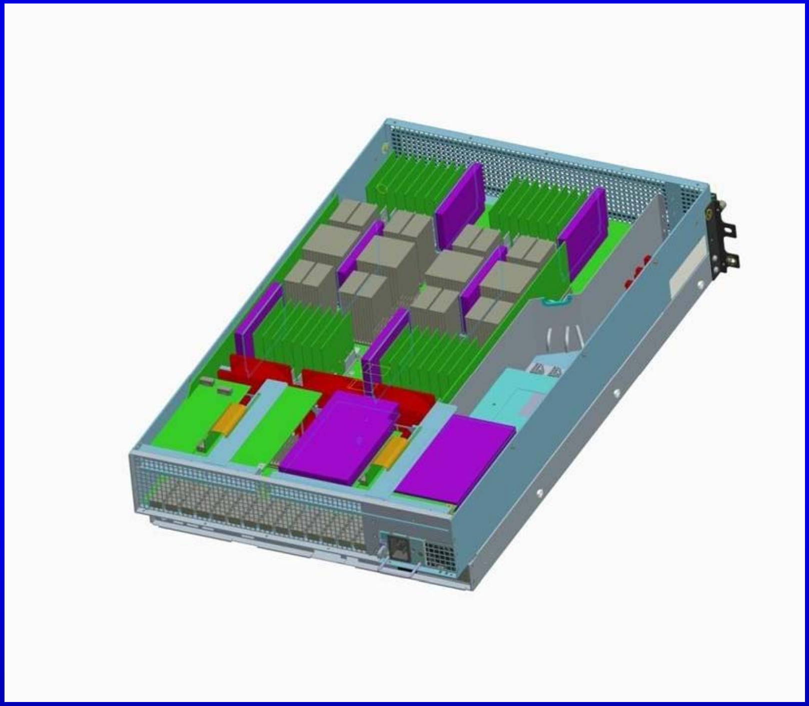
FPGA -Hard IP Floating Point

- Feature Set
  - By 2020
  - Fused Multiply Add – Mean Factor of 8 +/- 2
  - Reconfigurable IP Multiply and Add – Mean Factor 4 +/-1
  - Bigger fixed point DSP's – Mean Factor of 3
- Benefit
  - More Floating Point ALU's
  - More routing paths

CONVEY computer

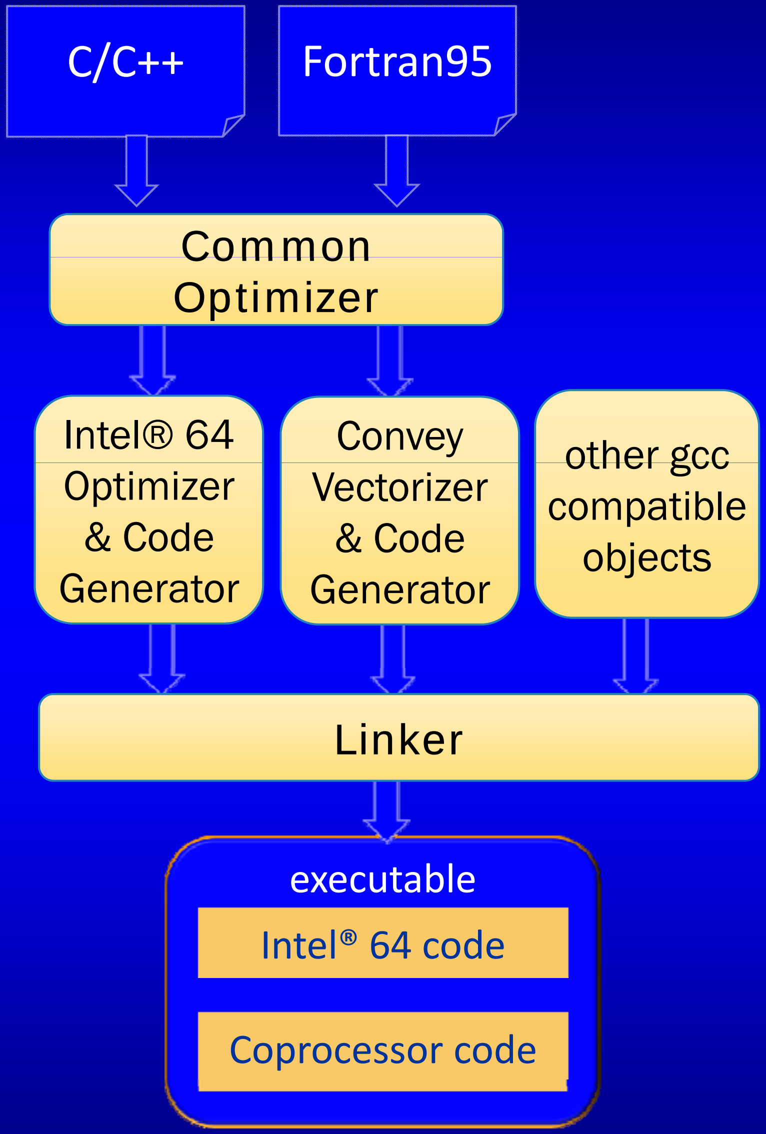
Number of AE FPGA's per node

- Feature Set
  - 4, 8, or 16 as a function of physical memory capacity
    - 4 -One byte per flop – mean factor of 1
    - 8 - ½ byte per flop – mean factor of 2
    - 16 -¼ byte per flop – mean factor of 4
- Benefit
  - More Internal Parallelism
  - Transparent to user
  - Potential heterogeneity within node



Convey Compilers

- Program in ANSI standard C/C++ and Fortran
  - PGAS ready
- Unified compiler generates x86 & coprocessor instructions
- Seamless debugging environment for Intel & coprocessor code
- Executable can run on x86_64 nodes or on Convey Hybrid-Core nodes


C/C++
Fortran95
Common Optimizer
Intel® 64 Optimizer & Code Generator
Convey Vectorizer & Code Generator
other gcc compatible objects
Linker
executable
Intel® 64 code
Coprocessor code

Programming Model

Example 4.1-1: Matrix by Vector Multiply

```
#include<upc_relaxed.h>
#define N 200*THREADS
shared [N] double A[N][N];    NOTE: Thread is 16000
shared double b[N], x[N];
void main()
{
int i,j;
/* reading the elements of matrix A and the
vector x and initializing the vector b to zeros
*/
   upc_forall(i=0;i<N;i++;i)
           for(j=0;j<N;j++)
          b[i]+=A[i][j]*x[j] ;
}
```

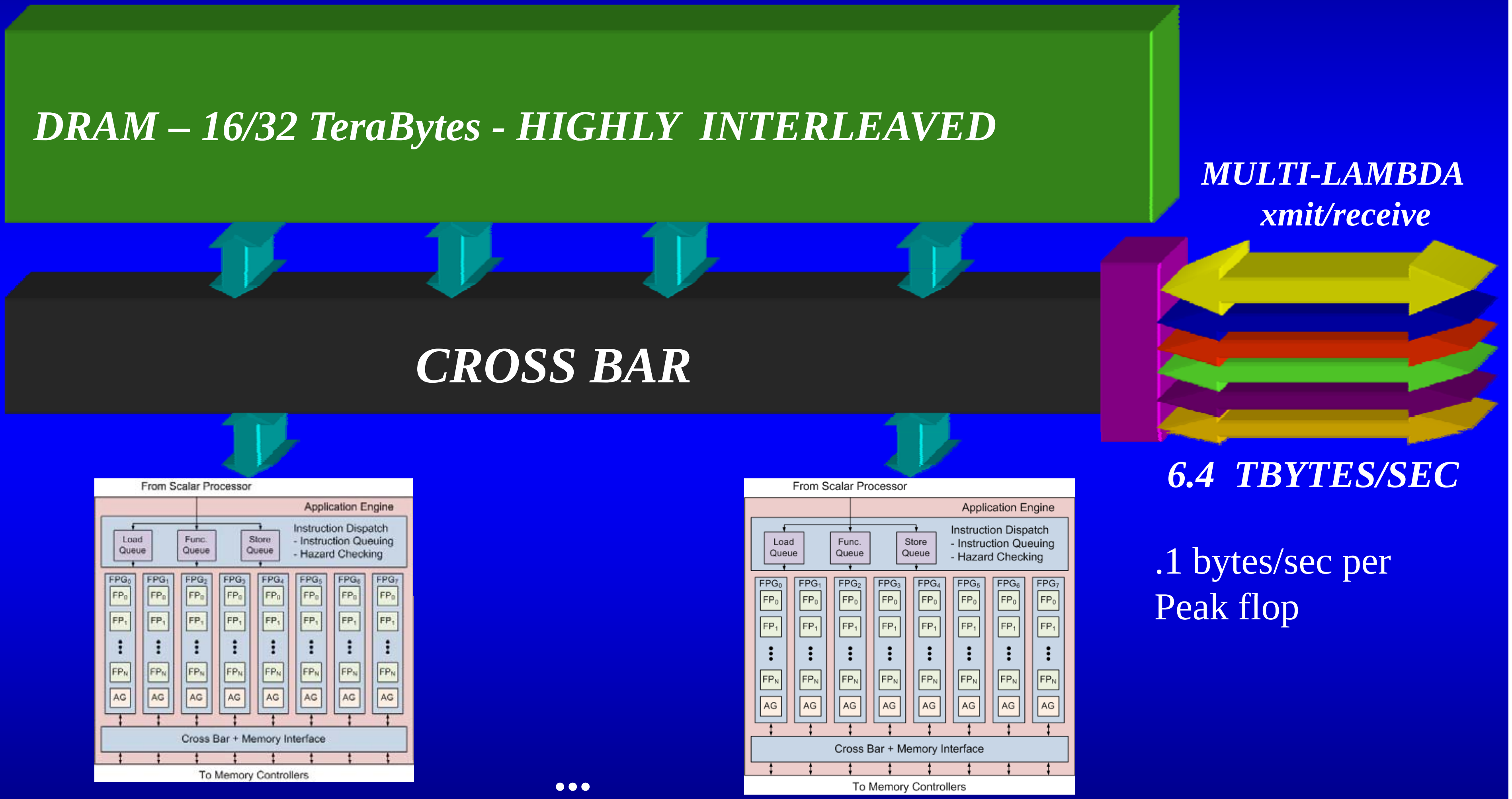
Math results in

- Using the mean
  - 7 Moore's law
  - 4 Matrix arithmetics
  - .90 efficiency (percentage of peak)
  - 8 Fused multiply/add (64 bits)
  - 4 16_ AE's (user visible pipelines) per node
  - Or a MEAN of 800 times today
    - Best Case – 2304
    - Worst Case - 448

The System

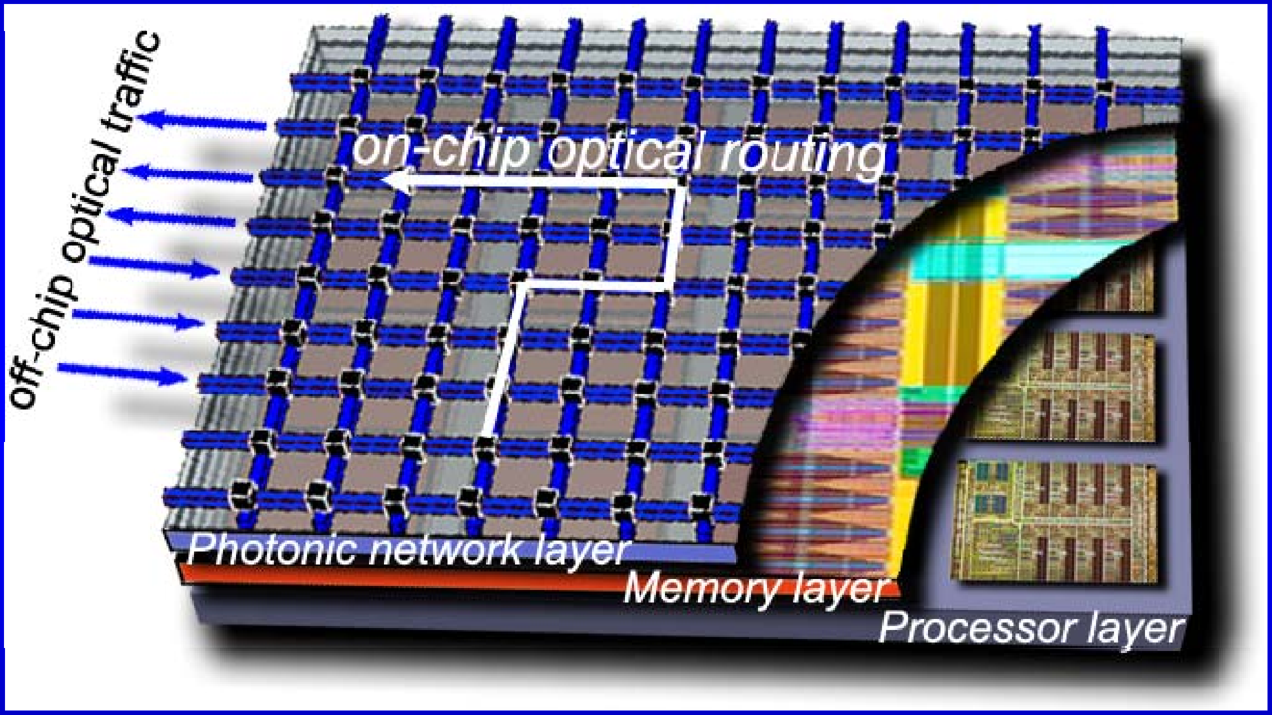
- 2010 base level node is 80 GFlops/node (peak)
- Thus 7 x 4 x .9 x 8 x 4 = 806 Factor
  - Mean of 800 = +1500 (upside)/- 400 (downside)
  - 64 TFlops/node (peak)
  - 16,000 Nodes/Exascale Linpack
- 64 bit virtual address space
  - Flat address space
  - UPC addressing paradigm integrated within TLB hardware
  - Programming model is 16000 shared memory nodes
    - Compiler optimizations (user transparent) deal with local node micro-architecture
- Power is 2 KWatts/Node (3U rack mounted)
  - 32 MegaWatts/system
  - 32 TBytes/Node (288 PetaBytes – system)
  - Physical Memory approx 60% of power

INTEGRATED SMP - WDM



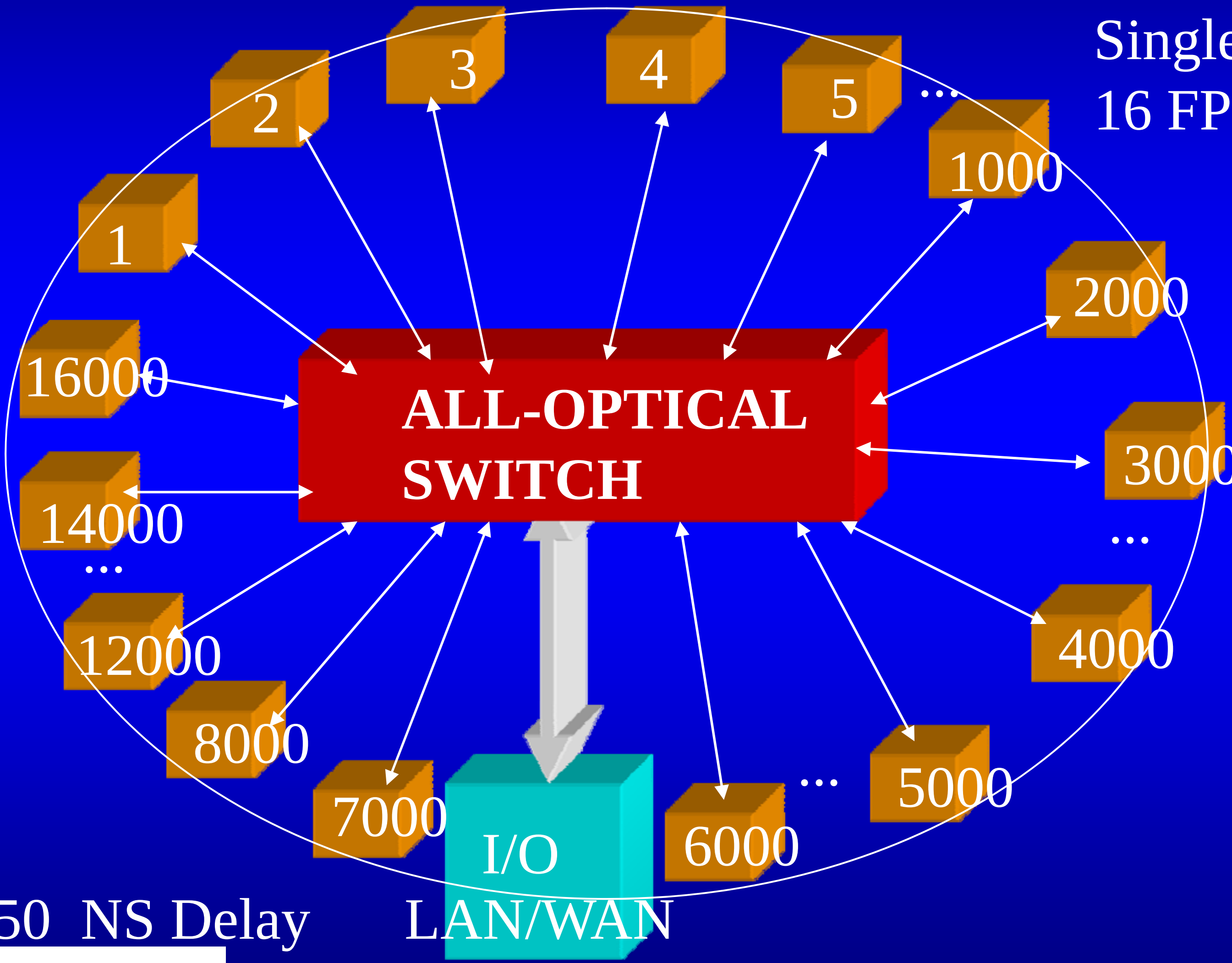
.1 bytes/sec per Peak flop

IBM Optical Technology



http://domino.research.ibm.com/comm/research_projects.nsf/pages/photonics.index.html

COTS ExaFlop/s System



Single Node
16 FPGA AE

10 meters= 50 NS Delay

What can be done to:

- Reduce power
  - 3D packaging
    - Reduce power in chip to chip interface
    - Better DRAM utilization
    - More power domains (selectively power up/down die regions)
- Architecture (relative to strawman)
  - Less Nodes
    - More matrix arithmetics
      - Twice the performance, same power
    - More floating point IP
      - Twice the performance, same power
  - Less Physical Memory
    - ¼ byte per flops or 15 TBytes/Node yields 20 MWatts
    - (144 PetaBytes Total)

Concluding

- Uniprocessor Performance has to be increased
  - Heterogeneous here to stay
  - The easiest to program will be the correct technology
- Smarter Memory Systems (PIM)
- New HPC Software must be developed.
  - SMARTER COMPILERS
  - ARCHITECTURE TRANSPARENT
- New algorithms, not necessarily new languages

YAHOO!
YEE HA!
TODAY'S MOTIVATIONAL SPEAKER: MR B. BILL
MURPHY
"He's working his way up to saying something inspirational."

CONVEY computer

Finally